Index

1. Introduction

2. ***Learning Finite-State Automata***

   - Using Positive Data only: ECGI, k-TSI, MGGI
   - Using Positive and Negative data: RPNI.
   - Probabilistic techniques.

3. Learning Finite-State Transducers

4. Applications

Regular Languages are Worth Inferring

- *Many practical problems admit a regular modeling* making the use of ”more powerful” recursive models unnecessary.

- Regular Languages can account for *local, short-term constraints* (like N-Grams) as well as for the more *global or long-term constraints* that often underlay in real aplications.

- Any language can be *approximated* (e.g. in a stochastic sense) *with arbritary precision* by a Regular Language.

- *Properties* of Regular Languages are relatively *well known*; this makes the development of inference methods easier.

- *Simple and efficient parsing methods* exist for strings belonging to Regular Languages.

Index

1. Introduction
2. Learning Finite-State Automata
   - ***Using Positive Data only: ECGI, k-TSI, MGGI***
   - Using Positive and Negative data: RPNI.
   - Using Probabilistic information.
3. Learning Finite-State Transducers
4. Applications

Error Correcting Grammatical Inference (ECGI)

[Rulot & Vidal, 87]

- ECGI is a grammatical inference heuristic: it was explicitly designed to capture the relevant ***regularities of concatenation and length*** exhibited by the substructures of unidimensional patterns.

- ECGI relies on ***error-correcting parsing*** both to build the grammars to be inferred and to deal with the errors (irregularities) of the patterns with respect to the learned grammars.

- ECGI builds up a ***stochastic regular grammar*** through a *single incremental pass* over the (positive) training set.

Stochastic ECGI

To achieve useful performance the inferred grammars must be complemented with statistical information:

- *Frequency of* utilization of each of the inferred *rules*.

- *Frequency of insertion deletion & substitution* of each symbol.

  Probabilities of both the error and non-error rules can be directly estimated from these frequencies, allowing *stochastic error-correcting parsing* to be used with new unknown samples.

If there are several classes with one grammar per class the parsing probabilities can be used for *maximum likelihood classification*.

Applications of ECGI

Speech Recognition:

- Speaker-Independent Spanish Digit Recognition [Rulot et al., 89]
- Language Modeling [Prieto & Vidal, 92]

Planar Shape Recognition (OCR):

- Mixed Size Font-independent printed digit recognition [Vidal et al., 92]
- Writer-independent Handwritten digit recognition [Vidal et al., 93]

Music processing:

- Learning Music Styles for automatic composition [Cruz & Vidal, 97]
- Music Style recognition [Cruz & Vidal, 98]

Banded chromosome recognition: [Vidal & Castro, 97]

k-Testable Languages in the Strict Sense (k-TS)

[García & Vidal, 90]

A k-TS Language is *defined by a four-tuple* $Z_k = (\Sigma, I, F, T)$ where:

- Σ is the ***alphabet***;
- I and F are sets of ***initial*** and ***final substrings*** of length smaller than k;
- T is a set of ***forbidden substrings*** of length k.

A language associated with Z_k is defined as [Zalcstein,72]:

$$L(Z_k) = I\Sigma^* \cap \Sigma^* F \; - \; \Sigma^* T \Sigma^*$$

$L(Z_k)$ consists of strings that *begin with substrings in I, end with substrings in F and do not contain any substring in T.*

Example:

$$Z_2 = (\{a,b,c,d,e\},\ \ \{a,d\},\ \ \{c,e\},\ \ \{a,b,c,d,e\}^2 - \{ab, db, bb, bc, be\})$$

$$L(Z_2) = \{abc, abe, dbc, dbe, abbc, abbe, dbbc, dbbe, abbbc, abbbe, dbbbc, dbbbe, \dots\} =$$
$$= (a+d)\, b^+ (c+e)$$

▷ ***Stochastic K-TS languages are equivalent to N-GRAM's with N=K [Segarra,93].***

k-TS Inference Algorithm (K-TSI)

[García & Vidal, 90]

Input : $k : \mathbb{N}$; $S : Set\ of\ strings$ //positive training sentences

Output : A_k=$(\Sigma, Q, \delta, q_I, Q_F)$ //Inferred Automaton

AuxVar : $x, y : Strings$; $q', q'', q : States$ //represented as strings over Σ

$\Sigma = \delta = \emptyset$; $q_I := \lambda$; $Q = \{q_I\}$; $Q_F := \emptyset$ //λ is the empty string

$\forall x \in S$ **do** $q' := q_I$;

  for $i := 1 \ldots |x|$ **do**

    if $\exists q'' \mid (q', x_i, q'') \in \delta$ **then** $q = q''$ //parse using current structure

    else //create new alphabet entry, state,

      $\Sigma := \Sigma \cup \{x_i\}$ //and/or transition, as required

      $y := q' x_i$; **if** $|y| > k - 1$ **then** $y := y_{2\ldots|y|}$ **endif**; $q := y$

      $Q := Q \cup \{q\}$; $\delta := \delta \cup \{(q', x_i, q)\}$

      if $i = |x|$ **then** $Q_F := Q_F \cup \{q\}$ **endif**

    endif

    $q' := q$

  endfor

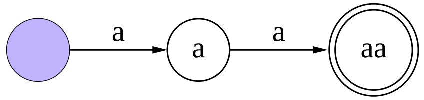
end$\forall$

$A_k := (\Sigma, Q, \delta, q_I, Q_F)$

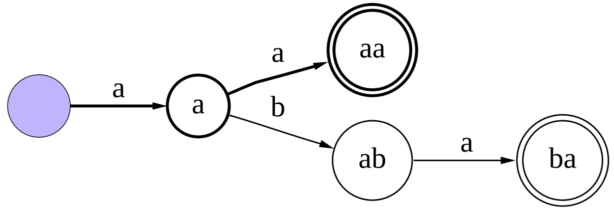
Illustration of k-TS Inference

Successive automata produced by k-TSI from $S = \{aa, aba, abba, abbba\}$ and $k = 3$.

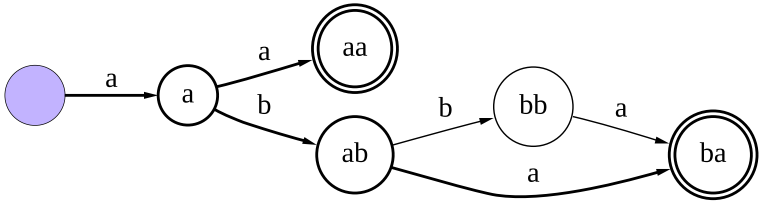
Thick lines represent states and transitions consolidated in previous steps, while thin lines are used for states and/or transitions that needed to be created in each step:



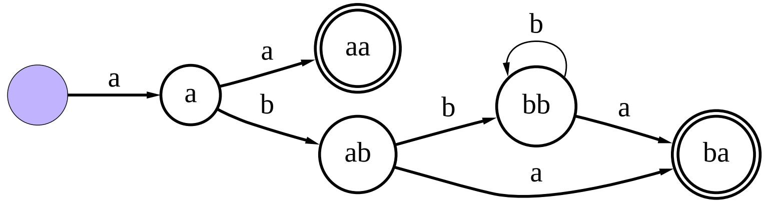
x=aa



x=aba



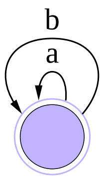
x=abba

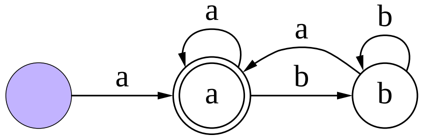


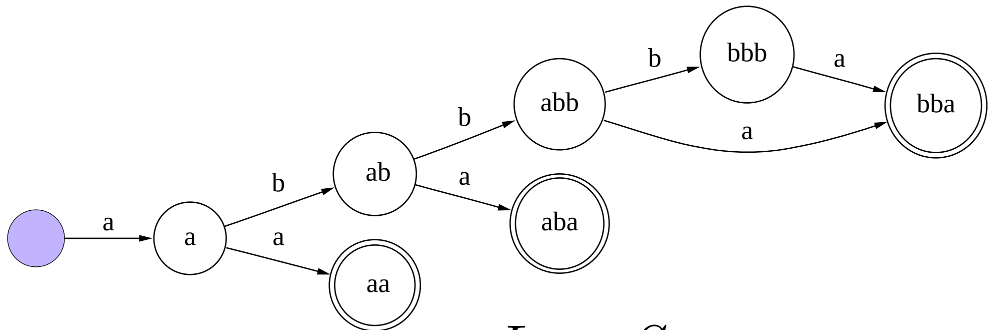
x=abbba

$$L_3 = ab^*a$$

Automata yield by k-TSI from $S = \{aa, aba, abba, abbba\}$ for $k = 1$, $k = 2$ and $k = 4$:







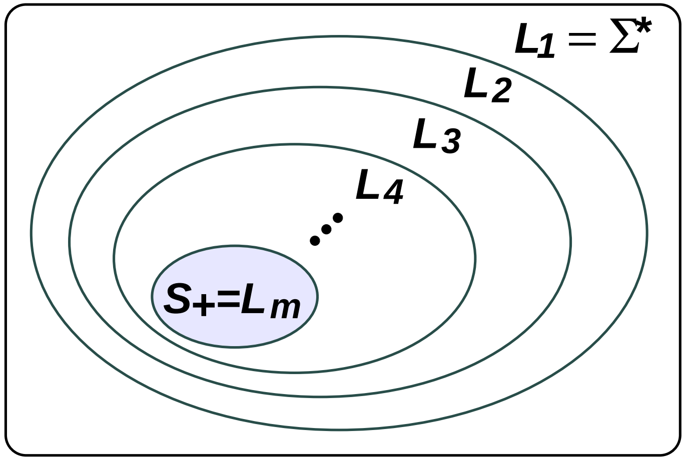
$$L_1 = (a + b)^* \qquad L_2 = a(a + b)^*a + a \qquad L_4 = S$$

Properties of k-TS Languages and the k-TSI Algorithm

[García & Vidal90]

Let $L_k(S)$ the k-TS language learned by k-TSI for a given sample S:

- $L_{k+1}(S) \subseteq L_k(S)$
- $L_m(S) = S, \quad m = max_{x \in S}|x|$
- $\forall S' \subset S \;\; L_k(S') \subseteq L_k(S)$
- $L_k(S)$ is the smallest k-TSL that contains S



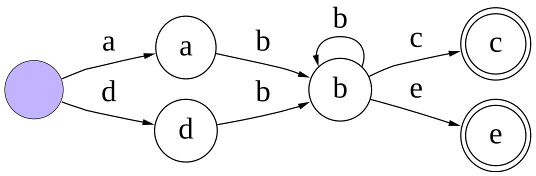
- For any fixed k the class of k-TS languages can be ***identified in the limit*** using the k-TSI algorithm with ***positive data***.
- The *whole class of Locally Testable Languages in the Strict Sense (LTS) can be identified in the limit* using k-TSI with *positive data* for increasing values of k and using *negative data* to control the growth of k.

The *LTS* class is the union of all k-TS languages for $k = 1, 2, 3 \ldots$

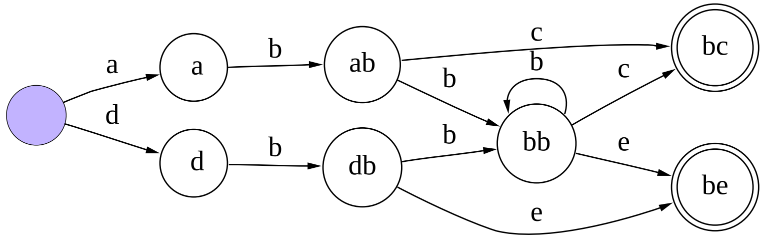
Limitations of k-TS languages

$$S = \{abc, dbe, abbc, dbbe, abbbc, dbbbe\} \quad \subset \quad (ab^+c) + (db^+e).$$

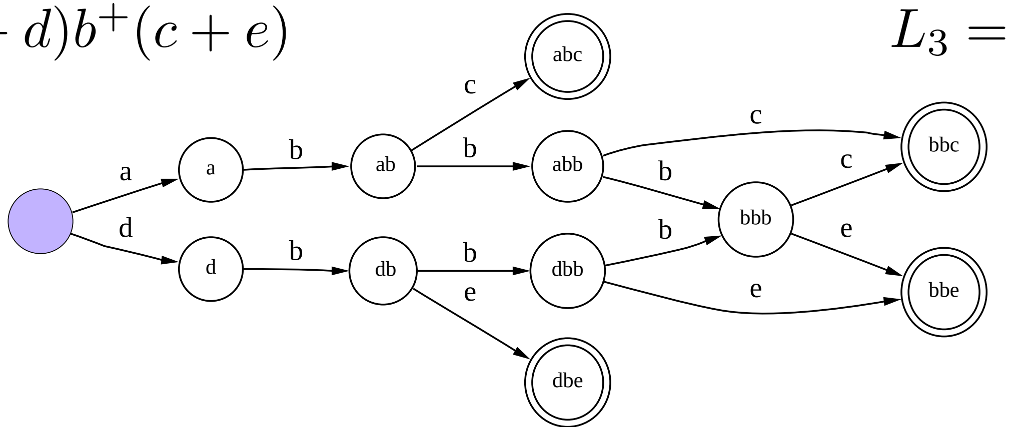
Automata yield by k-TSI for $2 \leq k \leq 4$:



$$L_2 = (a+d)b^+(c+e)$$



$$L_3 = L_2 - \{dbc, abe\}$$



$$L_4 = S$$

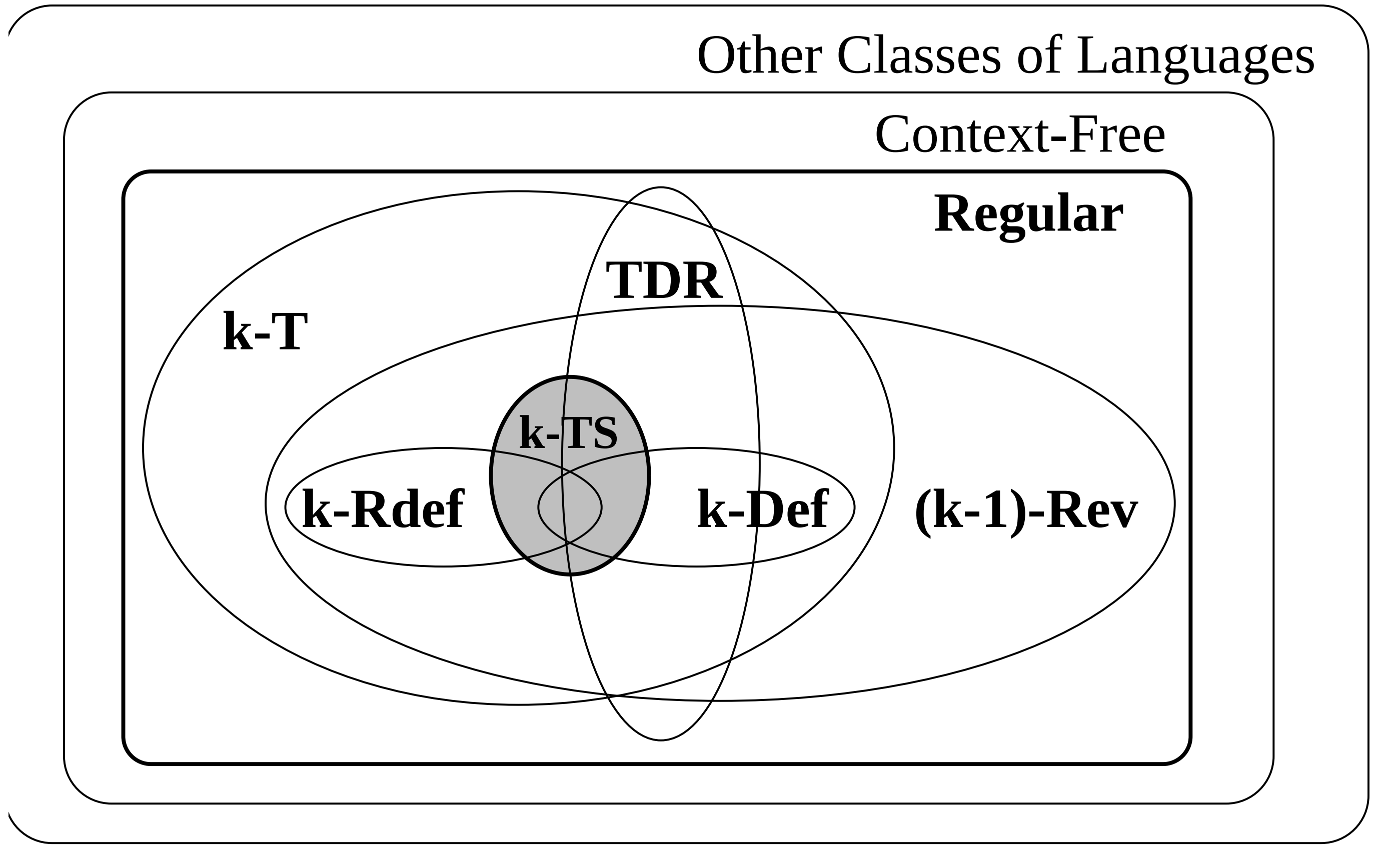
Inferred languages:

$$L_2 = (a+d)b^+(c+e) = \{abc, abe, dbc, dbe, \ldots, abbbbc, abbbbe, dbbbbc, dbbbbe, \ldots\}$$
$$L_3 = L_2 - \{abe, dbc\} = \{abc, dbe, abbc, abbe, \ldots, abbbbc, abbbbe, dbbbbc, dbbbbe, \ldots\}$$
$$L_4 = S = \{abc, dbe, abbc, dbbe, abbbc, dbbbe\}$$

L_2 and L_3 are clear overgeneralizations, while L_4 is exactly the training sample. No language is a satisfactory approximation to the target language.

Limitations of k-TS languages (cont.)

Some Families of Languages



K-TS languages are among the most restricted regular languages.

Even if we restrict ourselves to the class of Regular Languages (RL), many other possibilities exist that are significantly more powerful than k-TS/N-Grams, in the sense that they can help modeling *more global or long-term constraints*.

Morphisme-based Techiques: Morphisme Theorem

Any Regular Language can be Represented as a 2-TS language:

Morphisme Theorem [Medvedev,64]:

Let Σ be a finite alphabet and $L \subseteq \Sigma^*$ a regular language. There exist then a finite alphabet Σ', a letter-to-letter morphisme $h : {\Sigma'}^* \rightarrow \Sigma^*$, and a Local Language l over Σ' such that $L = h(l)$.

Example:
Let $L = \{1, 111, 11111, 1111111, \dots\}$ be the set of strings of 1's of odd length. L is (obviously) emphnot local; however it can be obtained by applying an alphabetic morphism h to the Local Language $l = l(Z)$:

- $Z = (\Sigma, I, F, T,) = (\{a, b\}, \{a\}, \{a\}, \{aa, bb\})$
- $l(Z) = \{a, aba, ababa, abababa, \dots\}$
- $h : \ \{a, b\}^* \rightarrow \{1\}^* : \quad h(a) = h(b) = 1$
- $h(l(Z)) = \{1, 111, 11111, 1111111, \dots\}$

A letter-to-letter morphisme between two alphabets Σ' and Σ is a function $h : {\Sigma'}^* \rightarrow \Sigma^*$ such that: $h(xy) = h(x)h(y) \ \forall x, y \in \Sigma'$; $\ h(\Sigma') = \Sigma$; and $h(\lambda) = \lambda$.

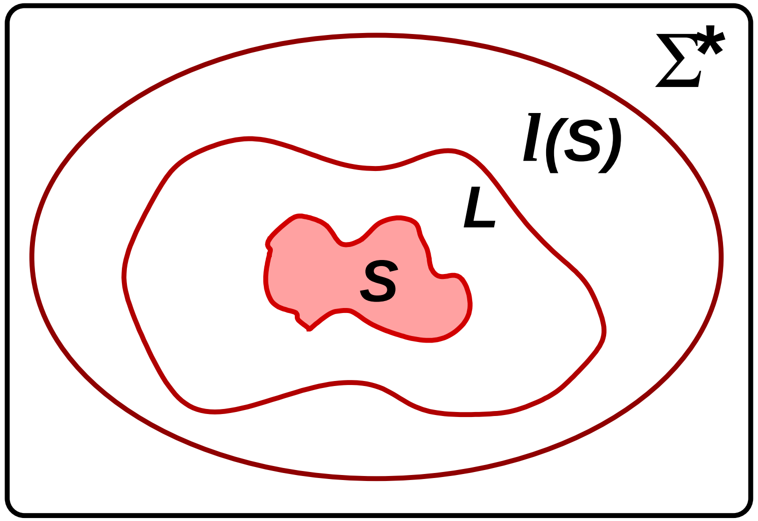
Learning General Regular Grammars from Positive Data: MGGI
[García et al.,87]

Morphic Generator Grammatical Inference (MGGI):

$$\begin{tikzcd} S \arrow[r, "g"] \arrow[d, "\mathrm{MGGI}"'] & S' \arrow[d, "\text{2-TSI}"] \\ L & l(S') \arrow[l, "h"] \end{tikzcd}$$

The lack of known target structure is compensated with a-priori knowledge about (perhaps long-term) syntactic constraints *that are desired to be captured by the inferred model. This knowledge is represented through appropriate word-renaming functions (*g *and* h*).*

Property [García et al.,87]: Let $S \subset \Sigma^*$ be a finite set of sentences and $L = h(l(g(S)))$ the language obtained from S by MGGI. If $h(g(S)) = S$, then $S \subseteq L \subseteq l(S)$.



MGGI: Example

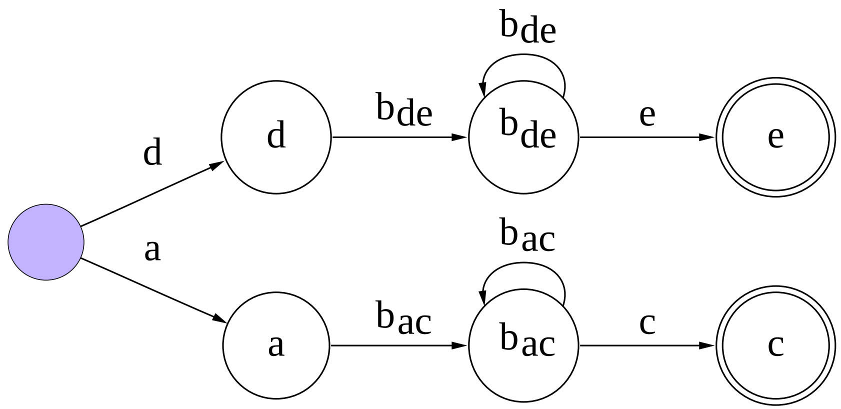
Let $S = \{abc, dbe, abbc, dbbe, abbbc, dbbbe\}$.

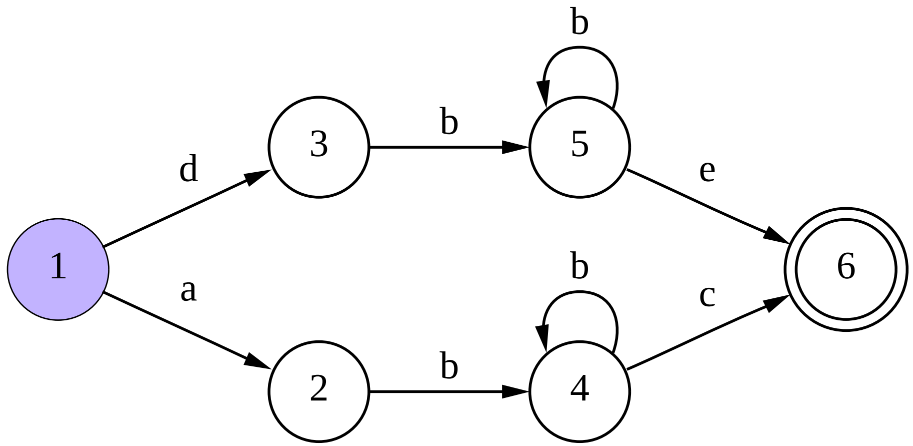
By inspection one can guess that a key syntactic feature consists of correctly matching beginings and ends of sentences. This suggests the following renaming function:

$$g(S) = S' = \{ab_{ac}c,\ db_{de}e,\ ab_{ac}b_{ac}c,\ db_{de}b_{de}e,\ ab_{ac}b_{ac}b_{ac}c,\ db_{de}b_{de}b_{de}e\}$$

Using S' as a training set, the 2-TSI algorithm yields the automaton on the left.

To comply with the condition of the MGGI Theorem (i.e., $h(g(S)) = S$) the morphisme h simply consists of droping the subindexes. By minimizing the result, the automaton on the right is obtained:





Applications of k-TSI and MGGI

Speech Recognition:

- Speaker-Independent Spanish Digit Recognition [García et al., 90] [Segarra, 93]
- Language Modeling [Vidal & Llorens, 96]

Music processing:

- Learning Music Styles for automatic composition [Cruz & Vidal, 97]
- Music Style recognition [Cruz & Vidal, 98]

Index

1. Introduction
2. Learning Finite-State Automata
   - Using Positive Data only: ECGI, k-TSI, MGGI
   - ***Using Positive and Negative data: RPNI.***
   - Using Probabilistic information.
3. Learning Finite-State Transducers
4. Applications

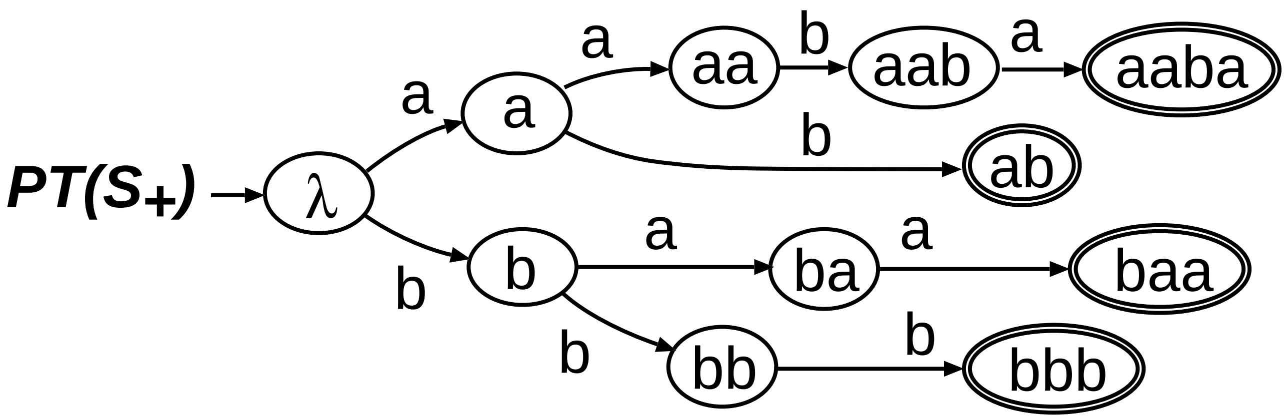
The Prefix tree Acceptor

- ***Set of Prefixes*** of a language: $L \subseteq \Sigma^* : \quad Pr(L) = \{u \in \Sigma^* | uv \in L,\ v \in \Sigma^*\}$

- ***Prefix Tree Acceptor*** of a finite set $S_+ \in \Sigma^* : \quad PT(S_+) = (Q, S, \delta, q0, F)$

  $Q = Pr(S_+); \;\; q0 = \lambda; \;\; F = S_+; \;\; \delta(ua) = ua$ iff $u, ua \in Pr(S_+)$

Example: $S+ = \{ab, aaba, baa, bbb\}$



Quotient Automaton or Automaton Derivative (A/π)

Let $A = (Q, \Sigma, \delta, I, F)$ and let $p = B_1, B_2, \ldots B_n$ be a partition on Q.

Quotient Automaton: $A' = A/\pi = (Q', \Sigma, \delta', I', F')$:

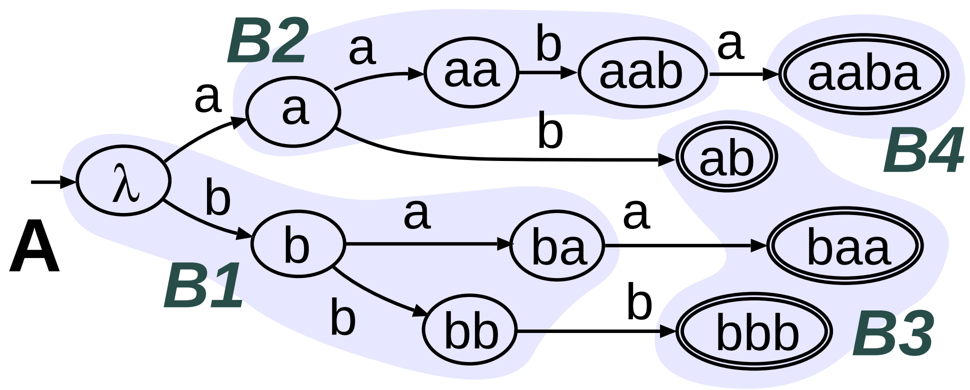
$$Q' = \pi, \quad I' = \{B_i \in \pi \mid B_i \cap I \neq \emptyset\}, \quad F' = \{B_i \in \pi \mid B_i \cap F \neq \emptyset\}$$

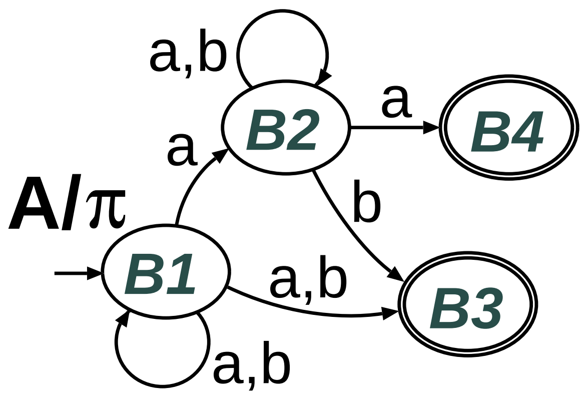
$B_j \in \delta'(B_i, a)$ if $q_i \in B_i,\ q_j \in B_j,\ q_j \in \delta(q_i, a)$

Example: $S_+ = \{ab, aaba, baa, bbb\};$

$$A = PT(S_+); \quad \pi = \{B_1, B_2, B_3, B_4\}, \quad I' = \{B_1\}; \quad F' = \{B_3, B_4\}$$

$$B_1 = \{\lambda, b, ba, bb\}, \quad B_2 = \{a, aa, aab\}, \quad B3 = \{ab, baa, bbb\}, \quad B4 = \{aaba\}$$





Properties of Prefix Tree Acceptor Derivatives
[Pao & Carr, 78] [Angluin,82]

Let $S_+ \subset \Sigma^*$ be a finite sample of a language L and $PT(S_+)$ its *Prefix Tree Acceptor*:

1. If $|\pi_1| < |\pi_2|$ (π_2 is finer than π_1) then $\mathcal{L}(PT(S_+)/\pi_2) \subseteq \mathcal{L}(PT(S_+)/\pi_1)$
2. If S_+ is ***structurally complete*** with respect to L then $\exists \pi : \mathcal{L}(PT(S_+)/\pi) = L$

Based on these properties different state-merging schemes lead to different GI methods. Two basic points of view:

- *Characterizable*: Choose a partition scheme that guarantees identification of a convenient class of languages. E.g.:
  - k-RI method for *k-Reversible languages* [Angluin,82]
  - General Regular Language Inference from + and - samples (**RPNI**) [Oncina,92]
- *Heuristic:* Choose a partition scheme that leads to generalizations of S_+ that are adequate for the aplication considered. E.g.:

  k-Tails [Bierman & Feldman, 72], Clustering of Tails [Miclet,80], k-Contextual [Muggleton,84]

Learning General Regular Languages from + and - Data

Given finite samples $S_+ \subset \Sigma^*$ and $S_- \subset \Sigma^*$, the problem of finding the *smallest Deterministic Finite Automaton* (DFA) A, such that $S_+ \subseteq L(A)$ and $S_- \cap L(A) \neq \emptyset$ is *NP-HARD* [Gold,78] [Angluin,78].

However we can instead try to obtain a DFA A' which is copmatible with S_+ and S_-, but *without insisting that the size of A' strictly be the smallest possible* for S_+ and S_-.

This idea has been followed in [Oncina,92], leading to the RPNI algorithm which has been shown to be able to (efficiently) identify any Regular Language in the limit using both $+$ and $-$ samples.

(Related approach: [Lang,92])

Learning General Regular Languages from + and - Data: RPNI Algorithm *[Oncina,92]*

```
Algorithm  RPNI  (Regular  Positive & Negative  Inference)
Input:     S+ S-
Output:    A: DFA which accepts S+ and do not accept R-

Method: A:=PT(S+); (let Q(A) denote the set of states of A)
        forall q in Q(A) - lambda in lexicographic order  do
           forall p < q in lexicographic order  do
               A'=merge(A,p,q)
               while A' is not deterministic  do
                    select q', q'' which violate determinism
                    A'=merge(A',q'q'')
               endwhile
               if A' accepts some strings from S-  then  A=A'
           end forall  p
        end forall  q
end  RPNI
```

Properties of the RPNI Algorithm [Oncina,91]

1. ***Correctness:*** the resulting automaton A is deterministic and $S_+ \subseteq L(A), \;\; S_- \cap L(A) = \emptyset$

2. ***Polynomial worst-case time complexity:*** $0(np^2 + p^3)$ where $n = \sum_{x \in S_-} |x|, \;\; p = \sum_{x \in S+} |x|$ (much better **linear** observed average cost)

3. ***Convergence:***

   - if S_+ contains a (small) *representative sample* of the unknown target language L then the resulting automaton A is the *smallest* DFA for L
   - using RPNI the class of Regular Languages can be identified in the limit from complete (both + and -) data with *polynomial update complexity*

Index

1. Introduction
2. Learning Finite-State Automata
   - Using Positive Data only: ECGI, k-TSI, MGGI.
   - Using Positive and Negative data: RPNI.
   - ***Using Probabilistic information.***
3. Learning Finite-State Transducers
4. Applications

Inference of Stochastic Regular Languages

- Stochastic Regular Languages can overcome Gold's negative computational results and can be effectively learned from only *positive data*; e.g. under Wharton's paradigm of approximate identification in the limit.

- The *lack of negative data* to control overgeneralization can be compensated by *statistical information gathered from the positive data*.

- Stochastic languages are particularly relevant for their use in most *real applications*

Learning Stochastic Regular Languages Through State Merging

- **Basic idea:**

  Given a finite sample S, orderly try merging the states of the Stochastic Prefix Tree Acceptor of S as long as the tails of the merged states have similar likelihood [Oncina,93].

- **Related approach:**

  If A is a current automaton, greedily merge those pairs of states of A which maximize Bayesian posterior probability $p(A|S) \sim p(S|A)p(A)$ [Stolcke & Omohundro,93]. The prior $p(A)$ is supplied by hand under the assumption that smaller and simpler models should have higher a priori probability.

Backward-Forward based techniques

- If an estimate of the appropriate number of states or non-terminals n is available, we can obtain a *locally optimal* estimate of the probabilities of a fully connected n-State Hidden Markov Model (HMM) from a sequence of training strings. Techniques to estimate the number of states n can be derived from [Ziv & Merhav,92]
- By (optionally) pruning out zero or low probability transitions a (stochastic) finite-state automaton can be obtained.
- A drawback of this technique is its high sensitivity to the probability initialization required by Baum-Welch/Backward-Forward reestimation [Stolcke & Omohundro,93]

Applications:

- Used to initialize the Inside-Outside algorithm for learning Context-Free Grammars [Lari & Young,90]
- Automata obtained by any other GI technique can be used to initialize Backward-Forward reestimation, generally leading to an increase of performance over the basic GI technique used [Casacuberta,90]